Using JPA Named Queries - Assignment

Assignment

Objective

In this project, you will build and extend the controller layer of a simple e-commerce system using **Spring Boot** and **Spring Data JPA**. This assignment focuses on implementing full **CRUD operations** along with **custom query methods** to handle product data stored in a relational database. The objective is to help you understand how to create RESTful APIs using Spring Boot and integrate them with JPA-based repositories.

You will be provided with a project template where the controller, entity, and repository files are blank or incomplete. Your task is to complete the implementation of the `ProductController`, define the JPA `Product` entity, and extend the `ProductRepository` interface with appropriate query methods. The emphasis is on using JPA repository interfaces to define meaningful data access patterns with minimal boilerplate code.

Project Setup

Your project structure is:
- **Main Application Class**: The main entry point of the Spring Boot application.
- **Controller Class**: A REST controller that will handle HTTP requests related to products.
- **Entity Class**: A JPA entity class that represents the `Product` entity in the database.
- **Repository Interface**: A Spring Data JPA repository interface to manage database operations for `Product`.

Key Components to Implement

1. Controller Class

In the `ProductController` class, you need to implement the following functionality:

- Class must have proper annotation to make it a rest controller.

- It must be mapped with "/api/products" request

- **Autowired Dependency**: The `ProductRepository` should be injected into the `ProductController` to interact with the database.

- **Method 1: Create Product (POST)**

  - Method Name: `createProduct`
  - Return Type: `Product`
  - Parameters: A `Product` object passed in the body of the HTTP POST request.
  - HTTP Method and Endpoint: Handles HTTP POST requests to the `/api/products` endpoint.
  - Functionality: Saves the product using the save method of the repository and returns the saved product.

- **Method 2: Update Product (PUT)**

  - Method Name: `updateProduct`
  - Return Type: `Product`
  - Parameters: Path variable `id` and request body `Product` object.
  - HTTP Method and Endpoint: Handles HTTP PUT requests to `/api/products/{id}`.
  - Functionality: Updates the product with the given ID by setting its ID and saving the updated object.

- **Method 3: Get All Products (GET)**

  - Method Name: `getAllProducts`
  - Return Type: `List<Product>`
  - Parameters: None
  - HTTP Method and Endpoint: Handles HTTP GET requests to `/api/products`.
  - Functionality: Returns a list of all products in the database.

- **Method 4: Get Product by ID (GET)**

  - Method Name: `getProductById`
  - Return Type: `Product`
  - Parameters: Path variable `id` (Long)
  - HTTP Method and Endpoint: Handles GET requests to `/api/products/{id}`.
  - Functionality: Returns the product with the given ID or null if not found.

- **Method 5: Delete Product (DELETE)**

  - Method Name: `deleteProduct`
  - Return Type: `void`

- Parameters: Path variable `id` (Long)
- HTTP Method and Endpoint: Handles DELETE requests to `/api/products/{id}`.
- Functionality: Deletes the product with the specified ID.

- **Method 6: Get Products by Category (GET)**

- Method Name: `getProductsByCategory`
- Return Type: `List<Product>`
- Parameters: Path variable `category` (String)
- HTTP Method and Endpoint: Handles GET requests to `/api/products/category/{category}`.
- Functionality: Returns a list of products matching the given category.

2. Entity Class

In the `Product` entity class, you need to implement the following:

1. **Entity Annotation**:
   Add the appropriate annotation to the class to mark it as a JPA entity, which will map to a database table.

2. **Named Query**:
   Define a named query on the class to retrieve products based on their category.

3. **Primary Key**:
   Add an annotation to the `id` field to mark it as the primary key, and set it to auto-generate its value using Identity technique.

3. Repository Interface

In the `ProductRepository` interface, you need to implement the following:
1. **Repository Annotation**: Ensure the interface is marked as a Spring Data JPA repository.
2. **Custom Query Methods**:
   - **Find by Category**: Implement a method to retrieve products by their category. The method should be named `findByCategory` and accept a `String category` parameter. This method uses a named query defined in the `Product` entity.

Execution Steps to Follow

1. **All actions like build, compile, running application, running test cases will be through Command Terminal.**
2. **To open the command terminal the test takers need to go to the Application menu (Three horizontal lines at left top) -> Terminal -> New Terminal.**
3. **cd into your backend project folder.**
4. **To build your project use command:**

   mvn clean package -Dmaven.test.skip
5. **To launch your application, move into the target folder (cd target). Run the following command to run the application:**

   java -jar <your application jar file name>
6. **To test any Restful application, the last option on the left panel of IDE, you can find ThunderClient, which is the lightweight equivalent of POSTMAN. Please use 127.0.0.1 instead of localhost to test rest endpoints.**
7. **Mandatory: Before final submission run the following command:**

   mvn test
8. **You need to use CTRL+Shift+B - command compulsorily on code IDE, before final submission as well. This will push or save the updated contents in the internal git/repository, and will be used to evaluate the code quality.**